We claim:

1. A multiprocessor based direct memory access (DMA) controller to carry out data transfers utilizing one or more transfer controllers, said controller comprising:

a first transfer controller supporting its own instruction thread; and

a second transfer controller supporting its own instruction thread and connected with the first transfer controller, whereby each transfer controller can operate as an independent processor or work together to carry out data transfers.

2. The DMA Controller of claim 1 further comprising:

a first DMA bus providing the first transfer controller with an independent data path to a plurality of local memories; and

a second DMA bus providing the second transfer controller with an independent data path to the plurality of local memories.

3. The DMA controller of claim 1 wherein both the first and the second controller are connected to a common system control bus (SCB) and a common system data bus (SDB).

4. The DMA controller of claim 2 wherein the local memories comprise sequence processor instruction memory, sequence processor data memory, and data memories for a plurality of processing elements.

5. The DMA controller of claim 1 wherein at least one of said transfer controllers further comprises a set of interconnected execution units including:

a core transfer unit;

an instruction control unit;

a system transfer unit; and

an event control unit.

6. The DMA controller of claim 5 further comprising an inbound data queue which is a data FIFO buffer written with data from a system data bus under control of the system transfer unit.

7. The DMA controller of claim 5 further comprising an outbound data queue which a data FIFO buffer written with data from a DMA bus under control of the core transfer unit, to be sent to a device or memory connected to a system data bus under control of the system transfer unit.

8. The DMA controller of claim 5 wherein the core transfer unit reads DMA instructions from a memory connected to the core transfer unit through a direct memory access bus.

9. The DMA controller of claim 5 wherein the system transfer unit fetches DMA instructions from memories or devices connected to the system data bus.

10. The DMA controller of claim 9 wherein a connection path allows DMA instructions fetched from the system data bus to be provided through an inbound data queue to the instruction control unit under control of the core transfer unit.

11. The DMA controller of claim 10 wherein the instruction control unit performs an initial decode of said fetching DMA instructions.

12. The DMA controller of claim 8 wherein the instruction control unit receives DMA instructions read by the core transfer unit from the memory attached to the DMA bus and the instruction control unit performs an initial decode.

13. The DMA controller of claim 5 wherein a path allows DMA instructions to be fetched from memories or devices on a system control bus through the event control unit and then forwarded to the instruction control unit.

14. The DMA controller of claim 2 wherein each transfer controller operates as a bus master and a bus slave on both the SCB and the SDB.

15. The DMA controller of claim 14 wherein each transfer controller as a bus slave on the SCB may be accessed by other SCB bus masters in order to read the internal state of each transfer controller or issue control commands.

16. The DMA controller of claim 14 wherein each transfer controller as a bus master on the SCB can send synchronization messages to other SCB bus slaves.

17. The DMA controller of claim 14 wherein each transfer controller as a bus master on the SDB performs data reads and writes from or to system memory or input and output devices which are bus slaves on the SDB.

18. The DMA controller of claim 14 wherein each transfer controller as a bus slave on the SDB can cooperate with another SDB bus master in a slave mode allowing the SDB bus master to read or write data directly from or to its data FIFO buffers.

19. The DMA controller of claim 1 wherein each transfer controller further comprises:

a transfer program counter (TPC) register; and

a wait program counter (WAITPC) register.

20. The DMA controller of claim 19 wherein the TPC and WAITPC registers have a sufficient numbers of bits to address all memories which may contain instructions.

21. The DMA controller of claim 19 wherein the TPC register stores the address of the next instruction word to be fetched and decoded.

22. The DMA controller of claim 19 wherein each transfer controller further comprises control logic operable to update a value of the TPC register after fetching a complete

instruction and to compare the value of the TPC register with a value stored in the WAITPC register.

23. The DMA controller of claim 22 wherein said control logic is further operable to suspend the fetching and decoding of instructions if the TPC and WAITPC values are the same.

24. The DMA controller of claim 19 further comprising a LOCK register and a set of LOCKID addresses which are recognized by each transfer controller to support mutually exclusive access to the WAITPC register by multiple host processors.

25. The DMA controller of claim 19 wherein instruction sequencing is controlled by executing branch-type instructions.

26. The DMA controller of claim 25 wherein said branch-type instructions include a jump-relative instruction which loads the TPC with the sum of the TPC and an immediate offset contained in the jump-relative instruction.

27. The DMA controller of claim 25 wherein said branch-type instructions include a jump-absolute instruction which loads the TPC with an immediate value contained in the jump-absolute instruction.

28. The DMA controller of claim 25 wherein said branch-type instructions include a call-relative instruction which copies an old value of the TPC value to a link counter register and loads the TPC with the sum of the old value and an immediate offset value contained in the call-relative instruction.

29. The DMA controller of claim 25 wherein said branch-type instructions include a call-absolute instruction which copies an old value of the TPC value to a link counter register and loads the TPC with an immediate offset value contained in the call-absolute instruction.

30. The DMA controller of claim 25 further comprising a link counter register storing a value, and wherein a return instruction copies the value of the link counter register to the TPC.

31. The DMA controller of claim 25 wherein said branch instructions employ condition specifiers which may be tested to determine whether a branch should be taken or not.

32. The DMA controller of claim 31 wherein at least one of the conditions specifiers is "always" meaning that the branch is always taken.

33. The DMA controller of claim 32 wherein at least one of the condition specifiers is arithmetic and selected from the group comprising equal, not equal, higher, higher or equal, lower, lower or equal, greater or equal, greater, less or equal or less.

34. The DMA controller of claim 31 wherein at least one of the condition specifiers is nonarithmetic and selected from the group comprising CTUeot, STUeot, NotCTUeot, NotSTUeot or always.

35. The DMA controller of claim 34 wherein the condition specifiers CTUeot, STUeot, NotCTUeot and NotSTUeot allow branches to be taken or not depending on transfer unit status.

36. The DMA controller of claim 31 further comprising a semaphore register storing a value and wherein the semaphore value is compared with zero, whereby if the relationship between the semaphore value and zero is the same as that specified by a condition specifier then the branch condition is true, and otherwise it is false.

37. The DMA controller of claim 1 wherein each transfer controller operates to fetch and decode transfer instructions sequentially in order to load transfer parameters into an appropriate execution unit.

38. The DMA controller of claim 37 wherein said transfer instructions include a flag bit in the instruction format which initiates concurrent execution of multiple transfer instructions.

39. The DMA controller of claim 37 wherein said transfer instructions include at least one of a transfer system inbound instruction, a transfer core inbound instruction, a transfer system outbound instruction, or a transfer core outbound instruction.

40. An instruction format for a transfer instruction comprising:

a base opcode field indicating that an instruction is of the transfer type;

a C/S field indicating that a transfer unit for the instruction is a core transfer unit or a system transfer unit;

an I/O field indicates whether the transfer direction is inbound or outbound;

a data type field indicates the size of each element transferred;

an address mode refers to a data access pattern which must be generated by the transfer unit;

a transfer count indicates the number of data elements of the size defined by the data type field are to transferred to or from a target memory or device before an end of transfer (EOT) occurs for that memory or device; and

an address field specifies a starting address for the transfer.

41. The instruction format of claim 40 further comprising an execute (X) field which when set to "1" indicates a start transfer event.

42. The instruction format of claim 40 further comprising an execute (X) field which when set to "0", parameters are loaded into a specified unit, but instruction fetching or decoding continues until a start transfer event occurs.

43. A method of transfer control operation comprising the steps of:

detecting a transfer controller reset event;

comparing a transfer program counter value with a wait program counter value;

updating either one of the transfer program counter value or the wait program counter value;

determining that the transfer program counter value and the wait program counter value are not equal; and

placing a transfer controller in a fetch state upon said determination.

44. The method of claim 43 further comprising the steps of fetching and decoding an instruction word.

45. The method of claim 44 wherein if an instruction comprises multiple words, repeating the step of decoding until all the multiple words are decoded.

46. The method of claim 44 further comprising the step of incrementing the transfer counter by one each time a word is fetched.

47. The method of claim 44 further comprising the steps of determining that a fetched instruction is a control type instruction; transitioning to an execute control state; and performing an action specified by the fetched instruction.

48. The method of claim 44 further comprising the steps of executing a wait type instruction and causing a transfer controller to transition to a wait state.

49. A multiprocessor system comprising a plurality of processors wherein at least one of the processors receives and decodes an instruction whose execution is controlled by a flag bit in the instruction format which if in an inactive state indicates no execution is to occur and which if in an active state indicates the concurrent execution of multiple processor instructions is to occur.

50. The multiprocessor system of claim 49 wherein if the instruction flag bit that controls its execution is inactive the processor waits for an appropriate event which triggers the instruction execution.

51. The apparatus of claim 49 wherein the multiprocessors comprise at least one core transfer unit and one system transfer unit operating with independent transfer counters making it possible to execute multiple transfer instruction in one transfer unit while the other transfer unit is processing a single transfer instruction.

52. A multiprocessor DMA system comprising: at least a first processor and a second processor to carry out DMA-to-DMA transfers between DMA controllers or I/O devices.

53. The multiprocessor DMA system of claim 52 wherein said processors employ a push model DMA-to-DMA transfer, each processor further comprising a transfer controller that is reading a data source which acts as a system data bus (SDB) master and writes data to an SDB slave address range of another transfer controller that is writing data to a destination memory.

54. The multiprocessor DMA system of claim 52 wherein said processors employ a pull model DMA-to-DMA transfer, each processor further comprising a transfer controller that is writing data to its destination memory which acts as an SDB master and reads data from an SDB slave address range of another transfer controller that is reading data from a source memory.